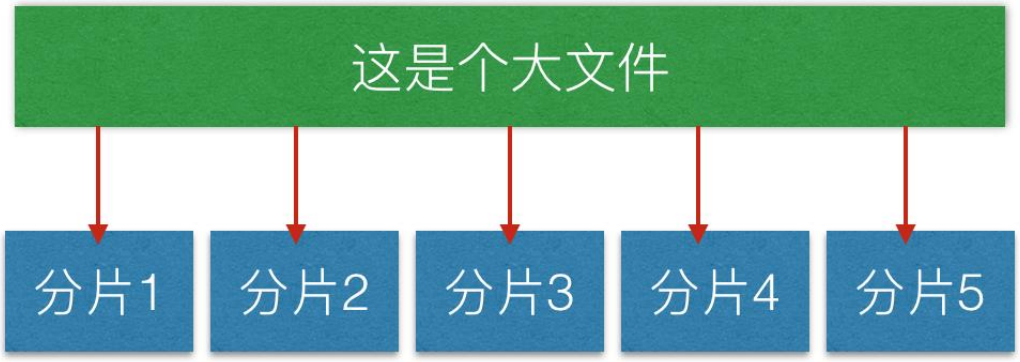
分片上传的核心思想。

一：什么是分片上传。

分片上传是把一个大的文件分成若干块，一块一块的传输。这样做的好处可以减少重新上传的开销。比如：
如果我们上传的文件是一个很大的文件，那么上传的时间应该会比较久，再加上网络不稳定各种因素的影响，很容易导致传输中断，用户除了重新上传文件外没有其他的办法，但是我们可以使用分片上传来解决这个问题。通过分片上传技术，如果网络传输中断，我们重新选择文件只需要传剩余的分片。而不需要重传整个文件，大大减少了重传的开销。

如下图是一个大文件分成很多小片段：



但是我们要如何选择一个合适的分片呢？

因此我们要考虑如下几个事情：

1. 分片越小，那么请求肯定越多，开销就越大。因此不能设置太小。
2. 分片越大，灵活度就少了。
3. 服务器端都会有个固定大小的接收 Buffer。分片的大小最好是这个值的整数倍。

因此，综合考虑到推荐分片的大小是 2M-5M. 具体分片的大小需要根据文件的大小来确定，如果文件太大，建议分片的大小是 5M，如果文件相对较小，那么建议分片的大小是 2M。

实现文件分片上传的步骤如下：

1. 先对文件进行 md5 加密。使用 md5 加密的优点是：可以对文件进行唯一标识，同样可以为后台进行文件完整性校验进行比对。
   2. 拿到 md5 值以后，服务器端查询下该文件是否已经上传过，如果已经上传过的话，就不用重新再上传。
   3. 对大文件进行分片。比如一个 100M 的文件，我们一个分片是 5M 的话，那么这个文件可以分 20 次上传。
   4. 向后台请求接口，接口里的数据就是我们已经上传过的文件块。(注意：为什么要发这个请求？就是为了能续传，比如我们使用百度网盘对吧，网盘里面有续传功能，当

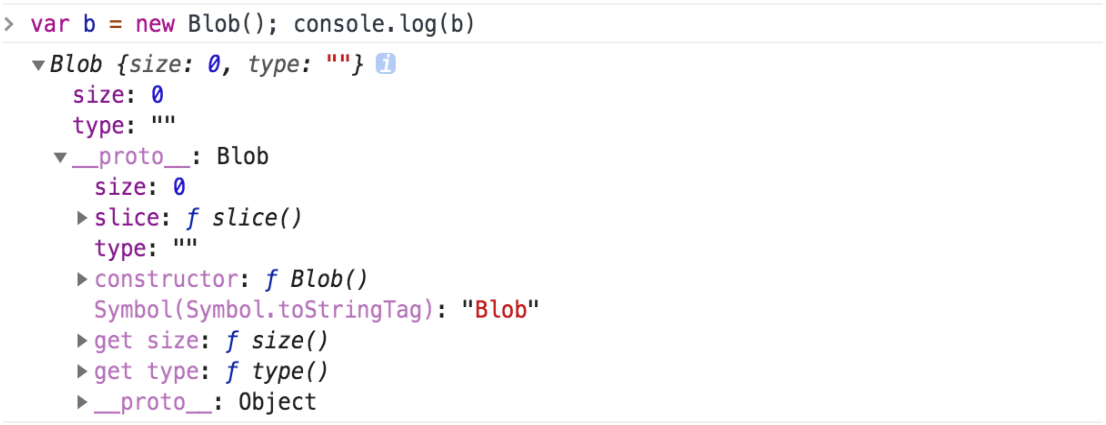
一个文件传到一半的时候，突然想下班不想上传了，那么服务器就应该记住我之前上传过的文件块，当我打开电脑重新上传的时候，那么它应该跳过我之前已经上传的文件块。再上传后续的块)。
5. 开始对未上传过的文件块进行上传。(这个是第二个请求，会把所有的分片合并，然后上传请求)。
6. 上传成功后，服务器会进行文件合并。最后完成。

二：理解 Blob 对象中的 slice 方法对文件进行分割及其他知识点

在编写代码之前，我们需要了解一些基本的知识点，然后在了解基础知识点之上，我们再去实践我们的大文件分片上传这么的一个 demo。首先我们来看下我们的 Blob 对象，如下代码所示：

```
var b = new Blob();
console.log(b);
```

如下所示：



如上图我们可以看到，我们的 Blob 对象自身有 size 和 type 两个属性，及它的原型上有 slice() 方法。我们可以通过该方法来切割我们的二进制的 Blob 对象。

2. 学习 blob.slice 方法

blob.slice(startByte, endByte) 是 Blob 对象中的一个方法，File 对象它是继承 Blob 对象的，因此 File 对象也有该 slice 方法的。

参数：
startByte: 表示文件起始读取的 Byte 字节数。
endByte: 表示结束读取的字节数。

返回值：var b = new Blob(startByte, endByte); 该方法的返回值仍然是一个 Blob 类型。

我们可以使用 blob.slice() 方法对二进制的 Blob 对象进行切割，但是该方法也是有浏览器兼容性的，因此我们可以封装一个方法：如下所示：

```
function blobSlice(blob, startByte, endByte) {
  if (blob.slice) {
    return blob.slice(startByte, endByte);
  }
  // 兼容 firefox
  if (blob.mozSlice) {
    return blob.mozSlice(startByte, endByte);
  }
  // 兼容 webkit
  if (blob.webkitSlice) {
    return blob.webkitSlice(startByte, endByte);
  }
  return null;
}
```

3. 理解 async/await 的使用

因此我们现在来看下如下 demo 列子：

```
const hashFile2 = function(file) {
  return new Promise(function(resolve, reject) {
    console.log(111);
  })
};
window.onload = async() => {
  const hash = await hashFile2();
}
```

如上代码，如果我们直接刷新页面，就可以在控制台中输出 111 这个的字符。为什么我现在要讲解这个呢，因为待会我们的 demo 会使用到该知识点，所以提前讲解下理解下该知识。

4. 理解 FileReader.readAsArrayBuffer()方法

该方法会按字节读取文件内容，并转换为 ArrayBuffer 对象。readAsArrayBuffer 方法读取文件后，会在内存中创建一个 ArrayBuffer 对象(二进制缓冲区)，会将二进制数据存放在其中。通过此方式，我们就可以直接在网络中传输二进制内容。
其语法结构：

```
FileReader.readAsArrayBuffer(Blob|File);
```

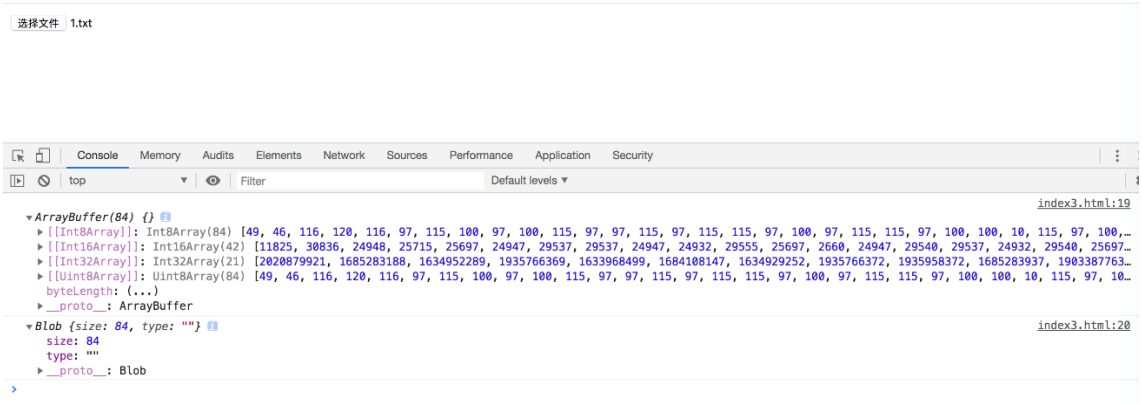
Blob|File 必须参数，参数是 Blob 或 File 对象。

如下代码演示：

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
<meta charset=" utf-8">
<title>readAsArrayBuffer 测试</title>
```

```
</head>
<body>
<input type="file" id="file"/>
<script>
  window.onload = function () {
    var input = document.getElementById("file");
    input.onchange = function () {
      var file = this.files[0];
      if (file) {
        //读取本地文件，以 gbk 编码方式输出
        var reader = new FileReader();
        reader.readAsArrayBuffer(file);
        reader.onload = function () {
          console.log(this.result);
          console.log(new Blob([this.result]))
        }
      }
    }
  }
</script>
</body>
</html>
```

如果我们现在上传的是文本文件的话，就会打印如下信息，如下所示：



三. 使用 spark-md5 生成 md5 文件

下面我们来理解下 上传文件如何来得到 md5 的值。上传文件简单的如下 demo，代码所示：

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>文件上传</title>
    <script src="./jquery.js"></script>
    <script src="./spark-md5.js"></script>
</head>
<body>
  <h1>大文件上传测试</h1>
  <div>
    <h3>自定义上传文件</h3>
    <input id="file" type="file" name="avatar"/>
    <div>
      <input id="submitBtn" type="button" value="提交">
    </div>
  </div>
  <script type="text/javascript">
    $(function() {
      const submitBtn = $('#submitBtn');
      submitBtn.on('click', async () => {
        var fileDom = $('#file')[0];
        // 获取到的 files 为一个 File 对象数组，如果允许多选的时候，文
件为多个
        const files = fileDom.files;
        const file = files[0]; // 获取第一个文件，因为文件是一个数组
        if (!file) {
          alert('没有获取文件');
          return;
        }
        var fileSize = file.size; // 文件大小
        var chunkSize = 2 * 1024 * 1024; // 切片的大小
        var chunks = Math.ceil(fileSize / chunkSize); // 获取切片的个
数
        var blobSlice = File.prototype.slice ||
File.prototype.mozSlice || File.prototype.webkitSlice;
        var spark = new SparkMD5.ArrayBuffer();
        var reader = new FileReader();
        var currentChunk = 0;

        reader.onload = function(e) {
          const result = e.target.result;
```

```
          spark.append(result);
          currentChunk++;
          if (currentChunk < chunks) {
            loadNext();
            console.log(`第${currentChunk}分片解析完成，开始解析
${currentChunk + 1}分片`);
          } else {
            const md5 = spark.end();
            console.log('解析完成');
            console.log(md5);
          }
        };
        function loadNext() {
          var start = currentChunk * chunkSize;
          var end = start + chunkSize > file.size ? file.size :
(start + chunkSize);
          reader.readAsArrayBuffer(blobSlice.call(file, start, end));
        };
        loadNext();
      });
    });
  </script>
</body>
</html>
```

如上代码，首先我在 input type = 'file' 这样的会选择一个文件，然后点击进行上传，先获取文件的大小，然后定义一个分片的大小默认为 2 兆，使用 var chunks = Math.ceil(fileSize / chunkSize); // 获取切片的个数 方法获取切片的个数。
如果 fileSize(文件大小) 小于 chunkSize(2 兆)的话，使用向上取整，因此为 1 个分片。同理如果除以的结果 是 1.2 这样的，那么就是 2 个分片了，依次类推.... 然后使用 SparkMD5.ArrayBuffer 方法了，详情可以看官网 (http://npm.taobao.org/package/spark-md5). 先初始化当前的 currentChunk 分片为 0，然后 reader.onload = function(e) {} 方法，如果当前的分片数量小于 chunks 的数量的话，会继续调用 loadNext()方法，该方法会读取下一个分片，开始的位置计算方式是：**var start = currentChunk * chunkSize;**
currentChunk 的含义是第二个分片(从 0 开始的，因此这里它的值为 1)，结束的位置 计算方式为：
var end = start + chunkSize > file.size ? file.size : (start + chunkSize);
也就说，如果一个文件的大小是 2.1 兆的话，一个分片是 2 兆的话，那么它就最大分片的数量就是 2 片了，但是 currentChunk 默认从 0 开始的，因此第二个分片，该值就变成 1 了，因此 start 的位置就是 var start = 1 * 2(兆)了，然后 **var end = start + chunkSize > file.size ? file.size : (start + chunkSize);**
如果 start + chunkSize 大于 文件的大小(file.size) 的话，那么就直接去 file.size(文件的大小)，否则的话，结束位置就是 start + chunkSize 了。最后我们使用

blobSlice 进行切割，就切割到第二个分片的大小了，blobSlice.call(file, start, end)，这样的方法。然后把切割的文件读取到内存中去，使用 reader.readAsArrayBuffer() 将 buffer 读取到内存中去了。继续会调用 onload 该方法，直到 进入 else 语句内，那么 const md5 = spark.end(); 就生成了一个 md5 文件了。